Information Processing Research Interim Technical Report

January 1979 to June 1979

Carnegie-Mellon University

DEPARTMENT OF COMPUTER SCIENCE

Information Processing Research
Interim Technical Report

January 1979 to June 1979

Carnegie-Mellon University
Computer Science Department
Pittsburgh, Pennsylvania 15213

Accession For	
NTIS GRA&I	X
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551.

Table of Contents

1. Introduction

This is the second semiannual report on Information Processing Research, supported by the Defense Advanced Research Projects Agency and monitored by the Air Force Avionics Laboratory under contract number F33615-78-C-1551. It covers the contract period January 1, 1979 to June 30, 1979.

This report is organized as six chapters under six major areas of research: Multiprocessor systems; Image Understanding systems; Machine Intelligence; Software Technology; Archival Memory Architectures; and Distributed Sensor Networks. Each chapter is self-contained and can be read without reading the entire report.

As can be seen from the report, the research efforts cover a wide spectrum of computer science and engineering. One of the strengths of the CMU environment is that the efforts are strongly interdependent and benefit from each other significantly. For example, efforts on Image Understanding systems benefit significantly from the effective use of Machine Intelligence techniques. The Software Technology and Multiprocessor programs strongly influence each other. The architecture research on multiple processor systems is appreciably affected by requirements of the Image Understanding program. Production Systems have found their way into many areas, including the compiler-compiler effort. Thus, work in all areas becomes important to the success of any of them.

In the remainder of the report we provide a brief description of the research in progress in each of the six areas supported under the present contract. Additional details are available in the reports described in the annotated bibliography at the end of each chapter.

2. Multiprocessor Systems

2.1 Introduction

Most large multiprocessor systems utilize rigid hardware structures that pre-allocate specific functions to specific processors. This CMU research group is developing alternate general purpose systems that will enable dynamic allocation of functions to processors, while also providing fault tolerance and graceful system degradation in the event of hardware failures. Such systems could conceivably run for long periods of time without a total failure. They would also provide a reliable hardware base for multilevel secure operating systems.

Cm* is an extensible, multiprocessor system with a hierarchical switching structure that in principle offers indefinite extensibility of processing power, memory capacity and communication bandwidth. The cost of its interconnection structure grows approximately linearly with system size. The addressing architecture presents to the programmer a uniform, system wide, segmented address space. Effective use of the structure of this system depends upon an object oriented, capability based operating system to support it and upon suitable decompositions of application which drive it (Swan 1978).

Since the first semi-annual report, the Cm* system has been successfully expanded from its 10 LSI-11 processor prototype to the complete 50 processor system. Multiple operating system and application program efforts have proceeded in parallel.

2.2 Hardware

During the past six months there were quite a few achievements in the hardware sector of the Cm* project. Design revisions were completed on the computer module backplanes, the initialization board and the local switch (Slocal) which connects the computer module with the interprocessor communication structure. These changes improved the manufacturability of the system and augmented the error recovery information preserved by the hardware, based on experience with the 10 processor version of Cm*.

Also finalized was the design of a 4K word (by 80 bit) writable control store for the Kmaps (inter- and intra-computer module communication controllers) which replaces the 1K boards previously used. This allows additional architectural experimentation and exploits recent static MOS memory developments.

Primarily used for Cm* operating system development, the link from Cm* to the department's PDP-KL10 has been installed. Both hardware and software aspects of this project are now functional. Five DA Links, used for high bandwidth connection between Cm*

and the CMUA, were constructed and tested over the past few months. CMUA's TOPS-10 operating system was modified to accommodate the link. User level programming was written to support primitive file transfers from the PDP-10 to Cm*. More sophisticated bidirectional transfer protocols have been defined and are partially implemented. At present, software development is continuing both on Cm* and on the PDP-10.

Several other things were accomplished along the lines of hardware expansion. As an aid to performance evaluation, the design of a 1 microsecond resolution clock with a battery backup was completed. Six used RP04 (80 megabyte) disk drives were made operational. The preliminary design of a custom disk controller for Cm* was drawn up. This new controller design offers a powerful yet inexpensive way of interfacing secondary storage into Cm*. The design emphasis is on providing a high level of error checking and fault recovery without sacrificing performance.

Unquestionably, the greatest hardware achievement in the past semester was the hardware expansion of Cm* from 10 processors to 50. The major steps which lead toward this goal are enumerated below:

1. Revised 10 existing computer modules.
2. Built and tested 40 new computer modules (includes backplane, modified LSI-11 processor, Slocal, initialization board).
3. Constructed two additional map bus monitors.
4. Revised 3 existing Kmaps and built 2 additional Kmaps.
5. Current system has 7.4 M bytes of primary memory.

The hardware development is now essentially complete. The only major outstanding item is the disk controller. Cm* should have at least five 80 megabyte disk drives interfaced one per cluster. There will be minor revisions of existing hardware and possible addition of primary memory and measurement facilities as dictated by the needs of the operating systems and applications. Also anticipated is continued experimentation with architectural features alterable in microcode.

2.3 Common Software

This section covers the development of utility software utilized by most of the research groups associated with Cm*.

Now that Cm* has expanded to 50 processors, the operating system, the Cm* Host, has

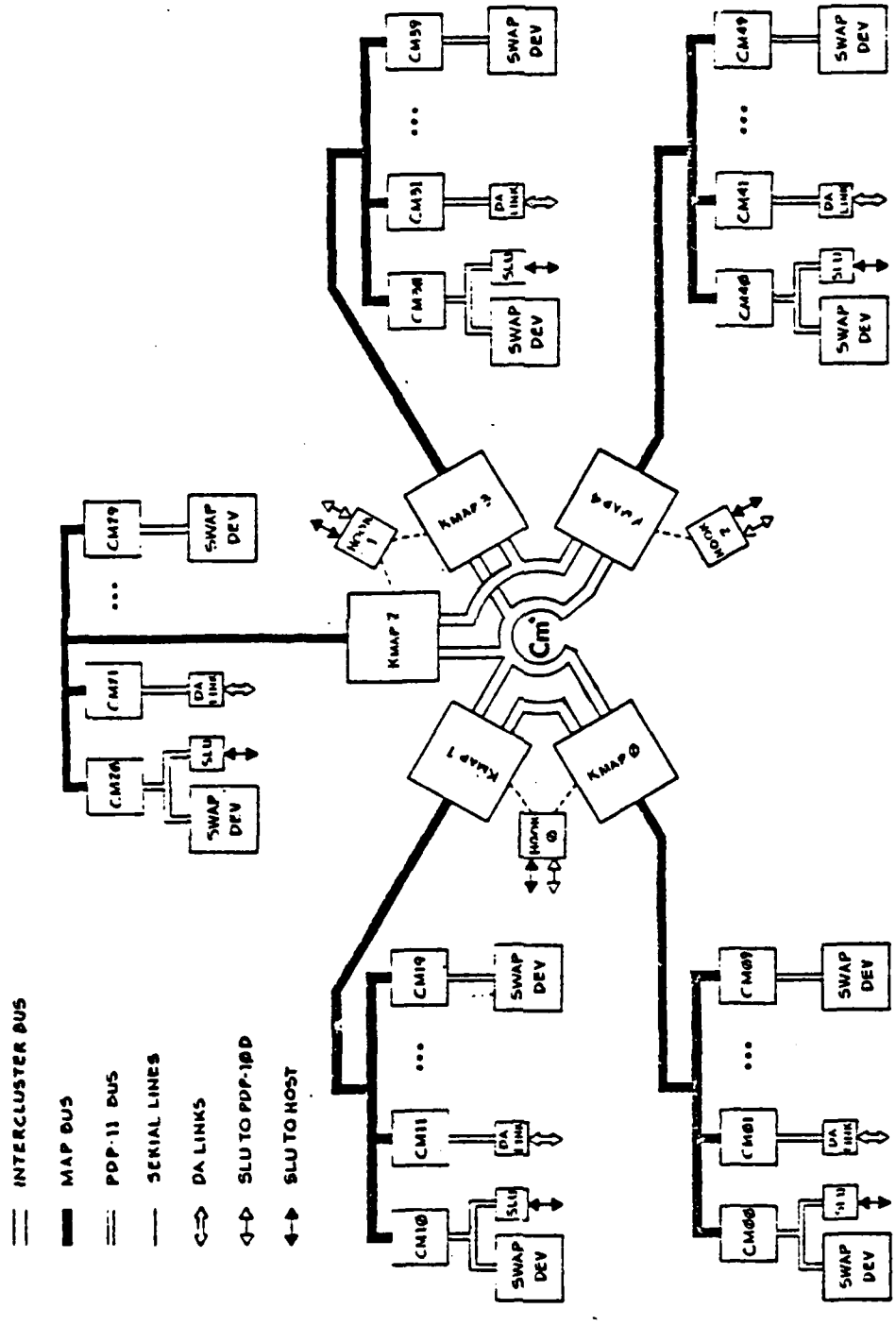


Figure 2-1: The Cm* hardware organization.

been reconfigured for this upgrade.

The Shepherd system, a file management and documentation system to aid development of large software systems, has stabilized. Now it is also used by non-Cm* related projects.

DA Link software support on the CMUA is now implemented and operational, as mentioned previously in the hardware section.

An operational microcode compiler, called Mumble, permits a programmer to write microcode for Cm* using high level constructs. An initial inspection shows that compiled code is at least 80% as efficient as if it were hand-generated.

Freshly implemented is a new Auto Diagnostic system for Cm*. This one automatically loads and runs diagnostics on computer modules which are not in use. Its presence has greatly assisted the staff in providing effective hardware maintenance. Also vital to this end are the hardware diagnostics, new or revised for: DA link, Slocal, Parity, disk and Kmap.

We are looking towards continued enhancement of existing software, particularly the enhancement of the file transfer system from the CMUA. A desirable new facility would be support of terminal interactions via the DA Link. This would allow full access to Cm* via the ARPAnet.

2.4 Multiprocessor Languages

Research in high level programming languages for multiprocessors previously led to an implementation of Algol68 on Cm* with a novel mechanism for detecting and exploiting parallelism at runtime. Present efforts concern development of a new language and operating environment which will adapt automatically to resource availability and provide error recovery.

The design of LO, a language based loosely on the Steelman Requirements for a common high order programming language, has been accomplished. It stresses a variety of process communication primitives to allow exploration of various methods; Ada is not a multiprocessor language, even though it has tasking facilities.

A compiler has been partially completed. It includes the following: parser, scanner, semantic routines (moderately modularized to accept table input), table generating routines. TCOL (tree based common language for compiler-compiler input and inter-phase communication) generation routines have also been designed and partially implemented. The intention is that when TCOL and Ada Runtime systems are mature, the compiler will be installed with them.

Implementation studies were conducted of multiple message sends and receives across a network, and of incremental garbage collection across a network.

The overall goal is an investigation of issues in providing an environment for developing large applications on a range of multiprocessor structures. The target application domain is image understanding, highly applicable at CMU.

LO implementation will wait until TCOL.Ada, the tree based common language for Ada, is finally designed, and until the Ada runtime system has stabilized. It now appears likely that the LO runtime system will have a greater similarity to the Algol68 runtime system, without automatic eventing, than it will be to the authorized Ada runtime system.

During the next semester, we will be working on a design and partial implementation of high-level language environment which addresses transparent handling of hardware configuration changes and provides good monitoring and debugging facilities.

2.5 The Medusa Operating System

This project is an investigation of the design of a fully distributed operating system based on a simple structure with low overhead and very fast message communications primitives. This is the second operating system for Cm*. The first system, StarOS, has been concerned with making Cm* programmable at a high level by users. Thus the StarOS system embodies a general object addressing mechanism and a set of tools for manipulating parallel programs. For Medusa we have chosen to emphasize problems of structure, rather than facilities.

The structure of Medusa results from a consideration of the questions of partitioning and communication in light of the three general goals (modularity, robustness, and performance) and their interaction with the architecture of Cm*. The resulting structure has two significant characteristics:

- The control structure of the operating system is distributed. The functionality of Medusa is divided into disjoint *utilities*. Each utility executes in a private protected environment, and implements a single abstraction for the rest of the system. Boundaries between utilities are crossed only by messages.

- Parallelism is implicit and expected in Medusa. All programs are organized as *task forces*, each of which is a set of cooperating *activities*. By making it the central unit of control, Medusa makes it possible for very fine-grain interactions to occur within a task force. Each Medusa utility is a single task force.

Over the past six months, this group completed the design and began the implementation of the operating system. Approximately 75% of microcode for Medusa has been written,

including a distributed name management system that is consistent, deadlock-free, and starvation-free without the use of a central authority.

Microcode has been newly designed to supply a message system that is an order of magnitude more efficient than existing systems: the cost of sending a message in Medusa is approximately the same as a high-level language procedure call.

Detailed design work was also done on the Medusa utilities. Actual programming of these was also initiated. Unfortunately, none of the code can be tested until the microcode is complete. The utilities include a memory manager, a file system that implements a Unix-like hierarchical structure, a task force manager, and an exception reporter that provides for cooperation among concurrent programs in handling exceptional conditions.

MACE is a recently designed and partially implemented debugging and measurement program oriented towards distributed environments like that of Cm*.

It is anticipated that the primary development and evaluation of Medusa will be completed within a year. Specific items include:

- Complete microcode implementation.
- Test utilities and debugger for Medusa.
- Transport simple Unix primitives including shell, C compiler, and file utilities.
- Experiment with error recovery to test out modularity and robustness of Medusa.

2.6 The StarOS Operating System

The object of the StarOS project is to produce an operating system of general utility for users of Cm*, and to provide a vehicle for experiments in program design and implementation for multiprocessors. StarOS is a capability based operating system that provides for multiple user *task forces*, each of which is composed of multiple processes executing concurrently. Each process or *environment* can address any memory of the machine uniformly, and all *objects* represented in the memory can be shared among the executing environments arbitrarily. StarOS provides for multiplexing the processors, communication among the various environments, and the transmission of information between Cm* and other resources, including user terminals.

In keeping with the recent Cm* hardware expansion, the one-cluster version of StarOS, a message-based operating system, has been expanded to multi-clusters. Also the microcode is

substantially complete and has been stand-alone tested using one cluster, in preparation for performing cross cluster operations. The microcode includes operations for capability addressing, as well as operations on stacks, queues, deques, and mailboxes.

The StarOS nucleus code, which executes in privileged mode, has been programmed and substantially tested on one cluster.

Among the goals for this group are to have, operational, a one-cluster StarOS system this summer and a multi-cluster StarOS system this year.

We also hope to determine the performance attributes of message-based operating systems, in contrast with monolithic (extant) systems.

Investigations of software structures for communicating process failure information will be initiated. We hope to determine whether the bailout mailbox mechanism, by which failing processes communicate failure data, provides an adequate mechanism to cope with the expected high failure rates of a multiprocessor structure.

We will also study whether capability addressing provides the b[illegible] for tailoring protected addressing domains so that each of the multiple processes involved in computation has access to only that data which it needs to know.

Also planned is experimentation with distributed memory allocation processes and a distributed garbage collection algorithm.

2.7 The TASK Specification Language

Distributed software, called task forces, is a collection of communication parallel processes that cooperate to solve an application problem. One of the substantial hurdles that must be overcome to utilize a multiprocessor effectively for large applications is that one must be able to specify and then maintain task forces that consist of a multitude of components. The specification language TASK has been designed to assist in such an effort.

TASK permits a task force author to incrementally specify the various modules that provide the various functions of the task force.

This specification is to be compiled to a sequence of commands to the StarOS loader which instantiates each of the various components of the task force, based on the creation and initialization parameters specified in the TASK description.

Task forces, by their very nature, vary more than uniprocessor software. For example, the number of partitions into which a data file needs to be partitioned varies with the amount of

data. The number of processes varies with the available resources.

TASK provides a high-level language in which to describe such dimensions of variability. For example, one may specify that the number of processes instantiated be the same as the number of computer modules in the cluster being used by the task force.

In addition, TASK provides the user with syntax to express resource usage directives, These directives constrain the allowable resource allocations. For the most part task force authors indicate the Proximity Relation between two components, whether the components are software or hardware. For example, one might state that two processes should be 'CmApart' so that they cannot compete for processor cycles.

Heuristics in the TASK compiler will accept such directives together with a current description of Cm* hardware and make resource allocation decisions, i.e. it will assign processors to processes, and it will place each object representation in some Cm memory.

A paper entitled 'TASK Forces: Distributed Software for Solving Problems of Substantial Size' has been accepted for presentation at the 4th International Conference on Software Engineering in the fall.

2.8 Applications

Due to the variety of departmental interests, and the work on several different operating systems for the multiprocessor, four unique application tasks have evolved. They are briefly itemized below:

Power systems

- A simulation of electrical networks consisting of passive devices (resisters, capacitors, and inductors) runs on the one-cluster version of the StarOS operating system.

- GOAL: measure performance aspects of simulation of power networks that include complex devices (e.g. power transmission lines) using a one- cluster system, then a multi-cluster system.

Monte Carlo Simulation of Molecular Movement

- Runs using 25 processors on a 25 particle water sample. The effective speed is 21 processors.

- GOAL: Determine where the equivalent of 4 processors are being expended, and improve performance of the simulation. We intend to expand the simulation to

involve more particles, more processors, and perhaps alternative simulation algorithms.

Molecular Orbital Calculations

- Some initial work, e.g. development of floating point library, has been completed.
- GOAL: Implement and evaluate representative stages of a molecular orbital calculation system that can be directly compared with existing implementations on CDC 7600s and DEC VAX/780s.

Image Understanding

- An initial study of how to decompose existing image understanding algorithms for Cm* has been made.
- GOAL: Implement an initial image understanding system by direct transfer and parallelization of existing PDP11/40 system.
- GOAL: When LO language system is available use it for a more elaborate implementation which will be fault tolerant and potentially transferable to other multiprocessor structures.

2.9 Annotated Bibliography

Fuller, S. H., Ousterhout, J., Raskin, L., Rubinfeld, P., Sindhu, P. and Swan, R. J. Multi-microprocessors: An overview and working example. *In* Computer Engineering: A DEC View of Hardware Systems Design. (C. G. Bell, J. C. Mudge and J. E. McNamara, Ed.) Digital Press, Bedford, MA, 1978. Also CMU-CSD Technical Report 1977.

An interesting phenomenon over the past several years has been the spontaneous growth of interest in multiple- microprocessor computer systems in many universities and research laboratories. This interest is not hard to understand given the inexpensive computational power offered by microprocessors today and the cost-performance improvements promised by those to be delivered in the near future. Microprocessors have had a dramatic impact on applications that require a small amount of computing. They have been used in instruments, industrial controllers, intelligent terminals, communications systems as special function processors in large computers, and more recent, in consumer goods and games. The

question naturally arises as to whether the microprocessor, which has proved so successful in these diverse applications, can be used as a building block for large general purpose computer systems. In other words, can a suitably interconnected set of microprocessors be used for tasks that currently require large uniprocessors capable of executing millions of instructions per second At present, there is no definitive answer to this question, but there are several reasons to believe that multiple-microprocessor systems might indeed be viable.

Siewiorek, D. P., McConnel, S. R. and Tsao, M. The measurement and analysis of transient errors in Digital computer systems. Proceedings of the 1979 International Symposium on Fault- Tolerant Computing, Madison, WI, June, 1979. Also CMU-CSD Technical Report May 1979.

Experimental data on transient faults from several digital computer systems are presented and analyzed. This research is significant because earlier work on validation of reliability models has concentrated only on permanent faults. The systems for which data have been collected are the DEC PDP-10 series computers, the Cm* multiprocessor, and the C.vmp fault tolerant microprocessor. Current results show that transient faults do not occur with constant failure rates as has been commonly assumed. Instead, the data for all three systems indicate Weibull distributions with decreasing failure rates.

3. Image Understanding Systems

3.1 Introduction

This research is developing automation aids for the interpretation of aerial imagery and is tied most closely to intelligence and cartographic requirements. The approach is to relate known apriori knowledge of an area to the photo being interpreted. This knowledge-based approach is applying current state-of-the-art techniques from the artificial intelligence field to the image interpretation problem. This work is also closely coupled to the parallel processing work at CMU. Image interpretation problems require the large number of low cost computer cycles promised by the evolving multiprocessor systems. The image domain also provides realistic constraints for the multiprocessor research task.

3.2 Overview of IUS Research

The primary objective of this research effort is to develop techniques and systems which will lead to the successful demonstration of image understanding concepts over a wide variety of tasks, using all of the available sources of knowledge. We are focusing our attention on three areas of research:

1. development of an integrated concept demonstration of an image understanding system

2. development and validation of concepts for computer architectures used in image understanding

3. development of interactive aids for tasks in image understanding

First, we are evolving an integrated concept demonstration of an image understanding system. The long-term goal of this research is to understand how knowledge can be used in the image interpretation process to produce systems which are 2 to 3 orders of magnitude more cost-effective than current systems. Over the next three years we expect to investigate how knowledge of maps, size and shape of landmarks such as buildings and rivers, and contextual relationships can be used in the interpretation of satellite images of the Washington, D.C. area and color scenes of downtown Pittsburgh.

The second area of research is the development and validation of concepts for computer architectures used in image understanding. The long-term objective of this research is to develop new computer architectures which will make low-cost image processing a serious possibility. We plan to evaluate the desirability of new processor designs and new instruction sets for image processing applications.

The third area is the development of intelligent interactive aids for tasks such as photo interpretation and map generation. Many of the same techniques which are useful in automatic interpretation are applicable in this area, except that in this case the human being provides the goal direction. The availability of intelligent assistants capable of examining large image data bases and retrieving desired information is expected to significantly improve human productivity in tasks such as photo interpretation and cartography.

The following is a brief summary of our work during this past semester.

3.3 Knowledge Representation and Search

During the past six months we have concentrated on the detailed performance analysis of the ARGOS Image Understanding System (Rubin, 1978) using hand segmented data. Problems with inaccuracies in the camera model used to generate the adjacency network and omissions in the knowledge network were identified and corrected. Further work is being done on mechanisms to resolve pointer conflict in the backtrace of the LOCUS search. These conflicts arise frequently due to the maintenance of multiple interpretations, each generated by the application of knowledge to different portions of the errorful signal data. One technique is to apply the Locus search technique to the backtrace to determine the "best" label.

We have begun to experiment with the use of a modified relaxation technique within the ARGOS Image Understanding System as an alternative to the LOCUS search. We use the same optical match, contrast, location, and adjacency knowledge producing results comparable to LOCUS but at a factor of 2 to 4 loss of speed. A report on this work is forthcoming in "An Experiment with Search Strategies for an Image Understanding System", (Smith, 1979).

3.4 Image Feature Analysis and Segmentation

A new approach to deriving three-dimensional surface orientation from image textural properties is described in "Shape from Texture: A Computational Paradigm" (Kender, 1979). Introduced is a new representational and computational tool, the normalized textural property map, which unites and exploits a large class of low-level image heuristics. An example of an application of the paradigm to an abstract textured image is given, and the relation of this work to existing work on shape is discussed.

We are continuing to study the effective use of knowledge in image segmentation. The KIWI segmentation program (Shafer and Kanade, in prep.) has incorporated a fast algorithm for extracting descriptions of regions resulting from a possible segmentation. The region extraction algorithm used in the KIWI program is being extended to perform other related

tasks. Shafer is using this procedure to eliminate noise regions, and has found it faster than "smoothing" techniques which accomplish the same task. The procedure is less sensitive to the size of the image than smoothing, and allows more flexible definitions of "noise". The phenomenon of "degenerate histograms" has been dealt with by the same algorithm, and we are able to identify "busy" (textured) areas of an image during segmentation without preprocessing. We have solved the problem of the large data tables required by this technique, and are extending it to gather additional statistics about the regions processed.

3.5 3-D Modeling

It is a common experience for us that, given a single 2-dimensional picture of an object, we have one (or a few) definite idea(s) about its 3-D shape, in spite of the fact that a large number of possible shapes exist which produce the same picture. This fact indicates that we use some assumptions or knowledge about the objects and about the image formation.

Kanade (Kanade, 1979) has been working to identify some of these assumptions, mostly in the geometrical aspects, by demonstrating how the theory and techniques which exploit such assumptions can provide a systematic shape-recovery method. The method consists of two parts: qualitative shape recovery and quantitative shape recovery. For the qualitative shape recovery we use a model of the Origami world (Kanade, 1978), together with edge profiles of *lines taken across the line in the image* in order to constrain line labels in the search of plausible interpretations.

For the quantitative shape recovery, we adopt a technique of mapping image regularities (in particular, the parallelism of lines and the skewed symmetry) into shape constraints, which is developed in (Kanade and Kender, 1979). Actual shape recovery from a single image is demonstrated for the scenes of an object such as a box and a chair. Given an image, the shape-recovery process generates a 3-D shape description of objects in terms of plane surfaces, and the description is supplied to a display program which can synthesize images of the same object as we would see it from other view directions.

3.6 Architectures for Image Processing

SPARC, the high speed processor being jointly designed by Control Data and CMU, has completed the design phase and begun layout and fabrication. Currently the four major component boards which contain the bulk of the custom LSI circuitry are being routed and fabricated. We expect that the processor will be delivered to CMU in the fall of 1979. Current gate level simulations indicate that instruction speeds in the order of 20ns can be expected.

Our collaboration with Texas Instruments (Eversole et al., 1978) to jointly design and develop an all-digital programmable VLSI chip set for several low level vision operations has begun to result in breadboard designs for several important operators: a programmable sum of products operator and a 5x5 median operator.

Work with our own Cm* group has produced an initial study of how to decompose existing image understanding algorithms. The goal is to implement an initial system by direct transfer and parallelization of the existing PDP11/40 system. When the LO language system is available, a more elaborate implementation which will be fault tolerant and potentially transferable to other multiprocessor structures will be attempted.

3.7 Annotated Bibliography

Reddy, D. R. Pragmatic aspects of machine vision. In Computer Vision Systems. (A. Hanson and E. Riseman, Ed.) Academic Press, New York, NY, 1978.

Over the last decade some aspects of machine vision such as edge detection, segmentation, and shape representation have received relatively more attention than other aspects of the machine vision problem. In this paper we examine some of the many different choices available to the designer of a machine vision system and discuss how lack of attention to any of these could seriously affect the research progress. We provide specific examples by drawing on current research in the CMU environment.

Our collaboration with Texas Instruments (Eversole et al., 1978) to jointly design and develop an all-digital programmable VLSI chip set for several low level vision operations has begun to result in breadboard designs for several important operators: a programmable sum of products operator and a 5x5 median operator.

Work with our own Cm* group has produced an initial study of how to decompose existing image understanding algorithms. The goal is to implement an initial system by direct transfer and parallelization of the existing PDP11/40 system. When the LO language system is available, a more elaborate implementation which will be fault tolerant and potentially transferable to other multiprocessor structures will be attempted.

3.7 Annotated Bibliography

Reddy, D. R. Pragmatic aspects of machine vision. In Computer Vision Systems. (A. Hanson and E. Riseman, Ed.) Academic Press, New York, NY, 1978.

Over the last decade some aspects of machine vision such as edge detection, segmentation, and shape representation have received relatively more attention than other aspects of the machine vision problem. In this paper we examine some of the many different choices available to the designer of a machine vision system and discuss how lack of attention to any of these could seriously affect the research progress. We provide specific examples by drawing on current research in the CMU environment.

4. Machine Intelligence

4.1 Introduction

This research is developing technology that is well suited to man/machine interaction questions. The rule-based systems technology (known as *Production Systems*) was invented at CMU as a cognitive psychology model for the human mind which is used in the computer to provide an interface to human thought processes. Rule-based systems have self-organizing properties that make them more flexible and easier to interact with than rigid tree programs. Instructable rule-based systems, a major thrust of the CMU effort, enables the user to modify the system without reprogramming. Another aspect of the CMU research is to build very large rule-based systems. That is, systems of the order of 3000 rules, as opposed to current technology which allows only 200 to 300 rules.

4.2 Production Systems

Research in production systems at CMU can be classifed as follows:

- Language and architecture design
- Efficiency of execution
- Possibilities and properties of very large production systems
- Study of how production systems can be built incrementally and automatically through natural-language-level interactions with an expert "instructor"
- Application of production systems to narrow problem solving domains, in order to better explicate the knowledge of that domain
- Exploration of the effectiveness of production systems as a candidate model for the mechanism underlying human cognition

Efforts and accomplishments in the last six months in the areas mentioned above by this research group are described below. For more details, the annotated bibliography at the end of this chapter will provide the titles of relevant technical reports and published papers.

RETE: algorithm and architecture for efficient implementation

The effort at CMU to improve the efficiency of production system interpreters has included several projects. One project is the study of the RETE Match Algorithm. In most previous production system interpreters, the productions (i.e., the statements in the program) were represented in the computer in a form that was isomorphic to the original productions. In this project a different approach was taken: a class of virtual production system machines

was defined and methods were developed for compiling the productions into the virtual machine language. The instructions for the virtual machines are similar to the instructions of a conventional von Neumann machine; they are quite unlike the productions in the original production system program. A recently published paper (Forgy 1979) details the latest results of this project. The important results include:

- The description of a particularly efficient virtual machine. The time cost of executing a production system on this machine varies with the logarithm of the number of productions in the system. The paper shows that (under some quite reasonable assumptions about the production system) this logarithmic time cost is the best attainable.

- Empirical and analytical studies of this class of production system interpreters. These studies determined how the time and space costs of interpreting production systems vary with the size of the production system and the amount of data processed by the system. Perhaps the two most important results are that the time cost varies with the logarithm of the size of the production system (provided the virtual machine is defined appropriately) and that the space cost varies directly with the size of the production system.

- The description of methods for implementing the virtual machine. It was shown that microprogramming could be used to obtain a speed increase of about 100 times over the interpreters currently used at CMU. An unexpected result was that while special hardware is certainly possible, it is not really necessary when these implementation methods are used.

Language design has been an evolutionary process. While the present languages are certainly more powerful and easier to use than earlier languages, there have been no sudden breakthroughs, only steady improvement. Another evolutionary step was taken this spring: OPS4, the most recent member of the OPS family of languages, was released. The first member of the OPS family of production system languages was developed at CMU in 1975, this language being based on several earlier CMU production systems.

IPMSL & XCON: computer configuration experts

Two of the issues that must be faced in order to develop an instructable production system are:

1. how to translate knowledge given to the system in (perhaps stylized) English into an appropriate set of productions (rules).

2. how to organize the knowledge in such a way that as more knowledge is added it does not interfere with (e.g., mask) knowledge that is already there.

Most of our work over the past year has addressed the second of these issues.

Two systems, IPSML and XCON, have been developed that can represent and manipulate computer descriptions (Rychener, McDermott). The knowledge in each system is highly modular and can be easily refined and extended. Though the work on XCON is currently being funded by CMU and Digital Equipment Corporation, it is a straight-forward application of the knowledge gained in earlier attempts to build instructable production systems. The IPMSL system of Rychener's represents and manipulates information at the PMS (processor-memory-switch) level; PMS is a high-level hardware description language invented by C. G. Bell and A. Newell. Its ultimate aim is to become an aid to design and analysis of computer systems. At the start of this period the system contained basic description, editing, and display capabilities; its size was 316 rules. A DEC VAX-11 computer description was formulated and entered into IPMSL as a test and demonstration of its capabilities. The system is quite general; VAX-11 is a typical medium-scale modern computer system, for which data was readily available. Information about computer components is represented as a semantic network, in which objects and their parts are described in terms of a set of attributes and values, and in which objects are interconnected by various sorts of relational links. The network representation is achieved directly within the production system architecture by a set of conventions, first developed in IPMSL, that allow fragments of the network to be stored as production rules. The system started out with an orientation towards closely interactive, natural-language instruction, but a shift has been made in strategy away from instruction issues and towards an exploration of the properties of large systems of rules, and in particular towards a system with the kinds of application that IPMSL has. As a result, rules are now entered directly in their complete form, and there has grown up a body of supporting software for managing this activity.

IPMSL has now grown to a size of 610 rules, with the addition of a number of new display, editing, inference, and data-checking capabilities. The system understands considerably more about the abstract attributes and values that are used to describe computers, and is able to interactively expand and correct that body of data as new computers are described. This new understanding has been applied to updating parts of the VAX-11 description that was initially entered to test the basic capabilities described above. The next step in developing IPMSL involves general methods for configuring (synthesizing) computer systems, given certain needs expressed by a user in terms of devices, memories, etc. While these methods are general, it is the intention to apply IPMSL to a specific configuration problem, that of configuring VAX-11s. This will involve adding a number of heuristic rules whose purpose is to criticize and guide the development of configurations by the general methods. That is, additional rules are needed to make sure that VAX-specific constraints remain satisfied. These specific critics can be added in a completely modular way, thus maintaining full generality of the underlying configuring methods, and leaving them open for application to

other domains. Note that IPMSL is a radically different approach from that of McDermott's XCON system. Information and assistance from the computer industry is central to this application. Another application of IPMSL that is getting under way also involves configuration, but of a different sort of object: the heat exchanger subsystems in chemical processing systems. This is a joint project, with a member of the CMU Chemical Engineering Department acting as a consultant.

The recent growth of IPMSL to a fairly large size relative to our PDP-10 computer has resulted in increasing slowness of operation, due to the heavy load of other jobs on the time-sharing system. This and other factors have led to the design and the start of implementation of a parallel processing version of the underlying production system language (OPS3) on CMU's multi-mini-processor, C.mmp. The design promises to be effective on a broad class of such computers, including CMU's Cm*. The initial implementation on C.mmp aims to confirm the basic properties of the design. The implementation is expected to take between three and six months, and some care will be taken to ensure that it is easily transferable to other computer systems.

4.3 ZOG Applications

ZOG is a rapid response, large network, menu selection system used for man-machine communication. ZOG is used in a number of task domains to explore and evaluate the limits and benefits of the communication philosophy. The rapid response and large network demands imply bandwidth requirements which stretch those available in commercial time-shared systems and architectures.

The papers in the bibliography by Mantei, McCracken and Robertson describe the system architecture, design considerations, and tools for building the information networks. In the past six months there have been two significant applications of the ZOG system relevant to our research efforts. The first is the **Ada management net** which we use for coordinating and tracking the progress on the Ada charette project (described in the Software Technology section). Any member of the group may update information within it relative to his own area of involvement. Any department member may retrieve information from it to arbitrary depth in any facet. The second is the **BROWSE library net** by Fox and Palay. This network permits on-line "browsing" access to the CMU Computer Science library as well as a parameterized approach to searching.

4.4 Heuristic Search

This research directed by Berliner has been principally concerned with how to structure evaluation functions that are to be used to discriminate desirability among elements of a very large set. The domain in this case is backgammon, however the techniques being used are general. Several very useful principles have been discovered concerning the use of non-linear functions in such a domain, resulting in evaluation functions that are quite sensitive to global conditions without being volatile. We have also identified two previously unknown effects, the Blemish Effect and the Suicide Construction, which have impeded the usefulness of previous work by other researchers. This research has resulted in the greatly increased strength of the backgammon program that we are developing. It is far stronger than any other such program and has been invited to participate in the World Backgammon championships in July 1979.

This group is also con'inuing analysis and experiments with the B* search algorithm and continue to find evidence that is is a very powerful search method, given sufficient knowledge support.

Gaschnig's work applies analysis of algorithm techniques to measure and analyze the performance of certain search algorithms and to the study of certain combinatorial problems. Some new algorithms are devised based on insights obtained from the performance evaluations.

4.5 Speech Understanding

Speech understanding research at CMU has been on-going for quite a few years now. This research has continued on a secondary level during the past semester. Several papers were published in this area; their abstracts may be found below.

We are also attempting to implement the Harpy speech recognition system on a special purpose computer architecture (i.e., multi-micro). Measurements of the current implementations (on PDP-10 and UNIX) have indicated that a great deal of the computer power required to execute the algorithm is used in storing and retrieving intermediate results. Therefore, it is reasonable to build a multiprocessor system using current (MOS) microprocessors and having:

- low cost
- equal or better performance (compared to the current systems)

More information may be found on the Harpy architecture in the technical report on the Harpy speech understanding system, whose abstract is below (Lowerre, 1976).

The Harpy algorithm has been decomposed such as to be runnable on multi-micro computers. It was restructured into sub-parts of equal complexity. At present, an operational simulator exists on the PDP-11, but no timings have been measured yet.

4.6 Annotated Bibliography

Berliner, H. J. The B* tree search algorithm; A best-first proof procedure. Artificial Intelligence 12 (1979).Also CMU-CSD Technical Report April 1978.

In this paper we present a new algorithm for searching trees. The algorithm, which we have named B*, finds a proof that an arc at the root of a search tree is better than any other. It does this by attempting to find both the best arc at the root and the simplest proof, in best- first fashion. This strategy determines the order of node expansion. Any node that is expanded is assigned two values: an upper (or optimistic) bound and a lower (or pessimistic) bound. During the course of a search, these bounds at a node tend to converge, producing natural termination of the search. As long as all nodal bounds in a sub-tree are valid, B* will select the best arc at the root of that sub-tree. We present experimental and analytic evidence that B* is much more effective than present methods of searching adversary trees. The B* method assigns a greater responsibility for guiding the search to the evaluation functions that compute the bounds than has been done before. In this way knowledge, rather than a set of arbitrary predefined limits can be used to terminate the search itself. It is interesting to note that the evaluation functions may measure any properties of the domain, thus resulting in selecting the arc that leads to the greatest quantity whatever is being measured. We conjecture that this method is that used by chess masters in analyzing chess trees.

Carbonell, J. G. Counterplanning strategies: Computer models of human reasoning in conflict situations. Proceedings of the First International Symposium on Policy Analysis and Information Systems, Durham, NC, June, 1979. Also CMU-CSD Technical Report February

1979.

A heuristic model of human reasoning is discussed where the reasoner must contend with dynamically-changing goals and actions of other actors in the world. A process model based on heuristic strategies is presented for decision making in obstructive and constructive counterplanning situations. The former situation is characterized by an actor striving to thwart the goals and plans of a second actor. The latter is the dual situation; it provides general means for an actor to pursue his goal in spite of attempts by others to block his plans. The model has been implemented as part of the POLITICS system, a computer program that understands simple natural language accounts of international political conflicts.

Forgy, C. L. On the efficient implementation of production systems. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, February, 1979.

It is not uncommon for an Artificial Intelligence program to spend most of its time evaluating patterns in order to locate either subprograms or entries in a data base. This is particularly true of production systems since, *unlike other programs*, they have no alternatives to pattern evaluation. Other programs may call some functions by name or access some data by retrieving the bindings of variables, but a production system uses pattern evaluation to select every procedure it executes and to locate every piece of data operated upon by the procedures. In this thesis, methods are described which can greatly reduce the amount of time that Artificial Intelligence programs like production systems spend in pattern evaluation. The thesis is concerned both with the algorithms used by a production system interpreter and with the hardware on which the algorithms are executed. The thesis contains a detailed description of a method for evaluating a set of patterns which (1) notes the similarities in the patterns so that it can avoid performing the same test more than once; (2) takes advantage of the fact that both the set of patterns and the set of objects change slowly by saving information from one evaluation to the next; and (3) allows a high degree of parallel activity during the evaluation. This method involves the use of a compiler which translates the patterns into a

program for a virtual pattern-matching machine. It is shown in the thesis that, although the instructions for this machine appear quite different from the instructions for a conventional processor, they can be interpreted efficiently on a conventional microprogrammed computer. If a microprogrammed computer were augmented with some inexpensive hardware described in the thesis, it would be able to interpret the virtual machine instructions as fast as it interprets conventional instructions. Without the special hardware, the computer would interpret the virtual machine instructions about three times more slowly. This thesis contains an analytical study of the pattern- matching algorithm and an empirical study of an interpreter which uses a Lisp implementation of the algorithm. These studies showed that the time required to evaluate the patterns would vary with the logarithm of the number of patterns. An interpreter running on a microprogrammed processor should be about two orders of magnitude faster than current interpreters; an interpreter running on the proposed special hardware should be about two and one-half orders of magnitude faster than current interpreters. The studies showed also that the space required to store the compiled patterns would be a linear function of the number of patterns. The compiled patterns would be smaller than the uncompiled patterns by a factor of perhaps two.

Gaschnig, J. Performance measurement and analysis of certain search algorithms. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, May, 1979.

This thesis applies the methodology of analysis of algorithms to study certain combinatorial problems and search algorithms originating predominantly in the AI literature, and extends that methodology to include experiments in a complementary role. Chapters 2 and 3 combine experimental and analytic techniques respectively to measure and to predict the performance of the A* best- first search algorithm, which solves path-finding problems defined in terms of finite strongly connected graphs. In this domain, we make numerous experimental performance measurements varying the heuristic function, the size of the problem, a weighting coefficient, and the performance measure; we derive general formulas in a simpler worst case analysis model that purport to

predict the experimental observations when evaluated at particular argument values that correspond to the experimental observations. The experiments use as case study a randomly generated set of instances of the "Eight" puzzle of varying size (depth of goal). The analysis in Chapter 3 extends the worst case tree search model of Pohl and others to arbitrary heuristic functions, resulting in cost formulas whose arguments include functions. Chapter 4 reports experimental results for a second problem domain, that of a class of satisfying assignment problems. Here we measure and compare under varying conditions the performances of four functionally equivalent algorithms -- the so-called backtrack algorithm, a version of the so-called "network consistency" or constraint satisfaction algorithm of Waltz, and two new algorithms BACKMARK and BACKJUMP. The experiments span four case studies: two sets of N-queens problems and two sets of randomly generated problems whose characteristics are specified by the values of certain parameters. Note that we are not interested primarily in the 8-puzzle or in the N- queens problems per se, but rather as relatively simple yet non-trivial case studies in which to explore general issues with rigor, principally the issue of predicting algorithm performance. The results take a number of forms: they variously confirm, disagree with or qualify hypotheses about algorithm performance found in the literature; tens of thousands of algorithm executions reveal new phenomena about algorithm performance; new algorithms are devised based on insights obtained from performance evaluation.

Lowerre, B. and Reddy, D. R. The Harpy speech understanding system. In Trends in Speech Recognition. (W. A. Lea, Ed.) Prentice- Hall, Inc., Englewood Cliffs, NJ, 1979. Also CMU-CSD Technical Report April 1976.

The Harpy connected speech recognition system is the result of an attempt to understand the relative importance of various design choices of two earlier speech recognition systems developed at Carnegie-Mellon University: The Hearsay-I system and the Dragon system. Knowledge is represented in the Hearsay-I system as procedures and in the Dragon system as a Markov network with a-priori transition probabilities between states. Hearsay-I uses a

best-first search strategy of the syntactic paths while Dragon searches all the possible syntactic (and acoustic) paths through the network in parallel to determine the globally optimal path. Hearsay-I uses segmentation and labeling to reduce the effective utterance length while Dragon is a segmentation-free system. Systematic performance analysis of various design choices of these two systems results in the HARPY system, in which knowledge is represented as a finite state transition network but without the a-priori transition probabilities. Harpy searches only a few "best" syntactic (and acoustic) paths in parallel to determine the optimal path, and uses segmentation to effectively reduce the utterance length, thereby reducing the number of state probability updates that must be done. Several new heuristics have been added to the Harpy system to improve its performance and speed: detection of common sub-nets and collapsing them to reduce overall phonemic types at every time sample, and semi- automatic techniques for learning the lexical representations (that are needed for a steady-state system of this type) and the phonemic templates from training data, thus automatically accounting for the commonly occurring intra-word coarticulation and juncture phenomena. Inter-word phenomena are handled by the use of juncture rules which are applied at network generation time, thereby eliminating the need for repetitive and time consuming application of phonological rules during the recognition phase. State transition probabilities are calculated dynamically during the recognition phase from speech dependent knowledge rather than a-priori from statistical measurements.

Mantei, M. M. and McCracken, D. L. Issue analysis with ZOG, a highly interactive man-machine interface. Proceedings of the First International Symposium on Policy Analysis and Information Systems, Durham, NC, June, 1979.

Policy decisions are made to resolve problems arising in complex, rapidly changing environments. This paper proposes a new type of information system, called ZOG, to analyze issues in such environments. ZOG is a large- network, rapid-response, menu-selection system which has some novel properties as a man-machine interface that make it a feasible tool for the

representation and management of large bodies of knowledge. The paper explores the development of a particular kind of ZOG network called an issue net: a hierarchically structured exposition of the arguments, evidence and counter-arguments associated with some policy issue. Two issue nets were built within ZOG, and one of these was used for a pilot study in public use of issue nets. The paper describes what was learned from this study and proposes further studies, both to substantiate ZOG's supposed benefits for policy analysis and to make progress on the problems with such uses of ZOG.

McCracken, D. L. and Robertson, G. Editing tools for ZOG, a highly interactive man-machine interface. Proceedings of the International Conference on Communications, Boston, MA, June, 1979.

The ZOG project at Carnegie-Mellon University is investigating a novel man-machine interface predicated on a large, rapid-response menu-selection network, with each node in the network being a display-screen-sized menu called a frame. This paper begins by introducing ZOG: its basic operation and its essention properties. Editing of frame networks (including their creation) has a major role in ZOG use due to the need for large networks. This is particularly true for an application called the ZOG Project Management Net, which is a large, shared frame network used as a communication medium by ZOG project members. Some initial data from Management Net use leads to observations on use of the ZOG editing tools. The existing ZOG editing facilities are then briefly described, followed by discussion of three separate approaches for obtaining advanced editing tools.

Rich, E. Building and exploiting user models. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, April, 1979.

This thesis addresses the problems that must be considered if computers are going to treat their users as individuals with distinct personalities, goals, and so forth. It first outlines the issues, and then proposes stereotypes as a useful mechanism for building models of individual users on the basis of a small amount of

information about them. In order to build user models quickly, a large amount of uncertain knowledge must be incorporated into the models. The issue of how to resolve the conflicts that will arise among such inferences is discussed. A system, GRUNDY, that builds, with the aid of stereotypes, models of its users, and then exploits those models to guide it in its task, suggesting novels that people may find interesting, is described. GRUNDY's performance is analyzed to provide some insight into how effective the user models are. The techniques that were developed for GRUNDY are shown to be appropriate for at least two other domains. The tradeoffs involved in designing such a user modeler for an arbitrary system are discussed. The issues involved in the modification of the data base of stereotypes to better describe the system's actual users is discussed. Some new questions raised by the ability to model individual users are raised.

Robertson, G. Some design considerations for the ZOG man-computer interface. Proceedings of the Third NATO Advanced Study Institute in Information Science, Chania, Crete, Greece, August, 1978.

This paper examines some of the issues in the design of man-computer interfaces. It discusses the evolution of these interfaces and presents a brief survey of the hardware and software mechanisms used to implement them. It discusses two particular interfaces in detail, including a discussion of their applications. The evolution of the technologies surrounding the computer has been dramatic during the last twenty years. However, it has not been at all uniform. The hardware designers, particularly for processors and memories, have clearly lead the way with continuous expansion of capability and reduction in cost, size, and power consumption. Software design has also gone through considerable evolution, but not nearly as rapidly. Although operating systems and programming languages are well understood, the programming methodologies in use still yield unpredictable results. The technology that has evolved most slowly has been the interface between the user and the computer. This is largely because few people have been willing to radically change the ratio between computing power spent on the interface and on the actual computation. In the early days, when processing power was a

scarce resource, the power spent on the interface was minimized. That has changed some, but not nearly as fast as the change to the processing power itself. This paper provides a brief survey of the hardware and software technology used in man-computer interfaces. The basic goals of these interfaces are discussed, including the nature of data transmitted, the interaction rate, ease of use, accuracy, and cost. A number of interactive input and output mechanisms are then presented. Output mechanisms discussed include linear text, split screen text, graphics, and audio. Input mechanisms discussed include typing, coded typing, pointing in several different ways, and speech. Each mechanism is discussed in terms of the general goals of man-computer interfaces. Two particular man-computer interfaces are discussed in detail. PROMIS is a medical information system being developed at the University of Vermont. It uses a novel interface based on a touch input, rapid response, menu selection system moving through a large network of knowledge states. ZOG is a general purpose interface being developed at Carnegie-Mellon University. It uses the same basic interaction philosophy as PROMIS, but is augmented with several features. It acts as a communications agent between a number of sources and destinations connected in a network. Its selections can perform arbitrary actions in addition to the normal function of moving through the knowledge base. It is also dynamic, in that it is user modifiable, and users are encouraged to adapt the system to their own needs. A number of applications of ZOG-like systems are discussed. These include guidance, instruction, control, management, and documentation. Finally, the experience of PROMIS and ZOG are discussed in relation to more global design issues for man-computer interfaces.

Yegnanarayana, B. Pole-zero decomposition of speech spectra. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, January, 1979.

A new method for determining the parameters of a pole-zero model for speech spectra is proposed in this paper. In this method the cepstral coefficients of a signal are split into two parts, one corresponding to poles and the other to zeros. The decomposition is achieved by using the properties of the derivative of phase

spectra of minimum phase signals. Parameters of the model are derived recursively from the cepstral coefficients for poles and zeros separately. Since poles and zeros are treated alike and derived independently, there is no effect of one on the other. The method is illustrated with several examples of speech spectra. It is shown that in all cases the envelope fit is equally good at peaks as well as at valleys in the spectrum. Results of this paper suggest a method of obtaining a linear system model for a given signal using a criterion different from the conventional minimization of mean squared error criterion. Although the method is described for minimum phase signals only, extension of the method to mixed phase signals is trivial, since a mixed phase signal can be split into minimum and maximum phase components using complex cepstrum.

Yegnanarayana, B. and Ananthapadmanabha, T. V. Design of digital filters by pole-zero decomposition. Technical Report, Carnegie- Mellon University, Computer Science Department, Pittsburgh, PA, February, 1979.

A new technique for design of digital filters based on a pole-zero model is proposed. The technique is capable of realizing any desired magnitude response specified in discrete frequency domain to the required degree of accuracy without significantly increasing the complexity for the computation of filter coefficients. It is shown that excellent characteristics for all types of the standard filters can be realized. The distinct features of the technique are: (a) Both the passband and the stopband have nearly flat characteristics; (b) A highpass (bandstop) filter can be realized as the reciprocal of a lowpass (bandpass) filter; (c) A high degree of flexibility exists in the choice of design parameters permitting trade-off between the realized response and the number of filter coefficients. The coefficients for the pole part and the zero part of the model are obtained in an identical manner using FFT algorithms and recursive relations, without the need for solving complex nonlinear equations. These features in the design are a result of application of a pole-zero decomposition technique for digital filter design. The technique, which was developed for modelling speech spectra, is based on the properties of the derivative of phase spectrum of a minimum phase signal.

Yegnanarayana, B. and Ananthapadmanabha, T. V. On improving the reliability of cepstral pitch estimates. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, February, 1979.

Identification of relatively high SNR regions in the short- time spectrum of a speech segment is very useful in speech processing applications. Such regions usually occur around the peaks in the spectral envelope. In this paper we propose a method for determining such regions automatically for a given speech segment. The method is based on a recently developed technique for pole-zero decomposition of speech spectra. It is shown that by selectively processing the high SNR regions of the spectrum, an unambiguous pitch peak in the high frequency portion of the cepstrum can be obtained. The processing involves computation of Hilbert envelope of the selectively filtered cepstrum. Several examples of speech segments are considered to illustrate the improvement provided by the proposed method.

Yegnanarayana, B. Speech analysis by pole-zero decomposition. Preprint of Papers on Speech Communication, Acoustical Society of America, Cambridge, MA, June, 1979. Also CMU-CSD Technical Report September 1978.

This paper describes a new method of speech analysis based on a pole-zero decomposition technique. The method overcomes the limitations of the widely used all-pole models for speech spectra. It is shown that a pole-zero model for the vocal tract system provides an accurate description of the spectral envelope. The model also provides a means of obtaining an accurate estimation of the source characteristics.

5. Software Technology

5.1 Introduction

This research is specifically concerned with obtaining "high quality" systems which are produced on time, which are produced to budget, which are correct, which are efficient, which can be maintained and enhanced, and which are tolerant to their human users. This effect emphasizes the previously neglected problems of generating optimized code for target machines whose instruction sets are symbolically described. The compiler-compiler and software verification efforts at CMU are contributing to DOD's goal of more cost-effective software tools and more portable software environments. These efforts can have a major impact on DOD software reliability.

5.2 Production Quality Compiler-Compiler

PQCC has concentrated on two areas during this past semester.

In the first set of work the structure and decomposition of the October compiler was carefully re-examined. The result was a greater understanding of the inter-relationship of the compiler phases. A list of the new compiler phases was then composed. Also developed were careful input-output specifications of the phases so that each phase would know what to expect from the previous phases, and would document what its output was for subsequent phases. This careful specification is intended to remove many of the interface misunderstandings that delayed the last compiler by several weeks.

Also investigated in depth were the issues of costing for code sequences. This is an important aspect, since it impacts not only the code generation, but also the assignment of result locations to physical machine locations. We introduced here the concept of Temporary Name TNs which are records used to hold all the information about the results of the register allocation that must be available to the code generator. They are used to represent entities that must be allocated storage, such as variables or the results of expressions. The TNASSIGN phase determines the correspondence between TNs and program entities. A subsequent phase is packing which establishes the correspondence between TNs and target machine storage locations. The evaluation centers primarily around the phases known as TNASSIGN and PACK (see previous reports).

The code generator was completely redesigned due to the redesign of TNASSIGN and PACK. The current design has the particular property that it will choose, whenever possible,

the actual lowest-cost code sequence, where costing is a complex function based on the "cost" of an instruction (i.e., in time, space) and the total amount of the target tree it absorbs. The search strategies are such that no backup is required in the search. The input-output specifications of the code generator have been finished except for minor revisions, but implementation will not start until the optimizing Ada compiler effort begins in July.

A tree-transformation language was designed, discussed, and is currently being modified in some minor ways. This will be used to implement many of the phases of the PQC which are implemented as tree transformations. The code generation implementation was largely influenced by the work described in Cattell's thesis on the formalization and automatic derivation of code generators for the optimizing compiler. He provides an explanation of TCOL (tree based common language) and its application as a base for the derivation of translators.

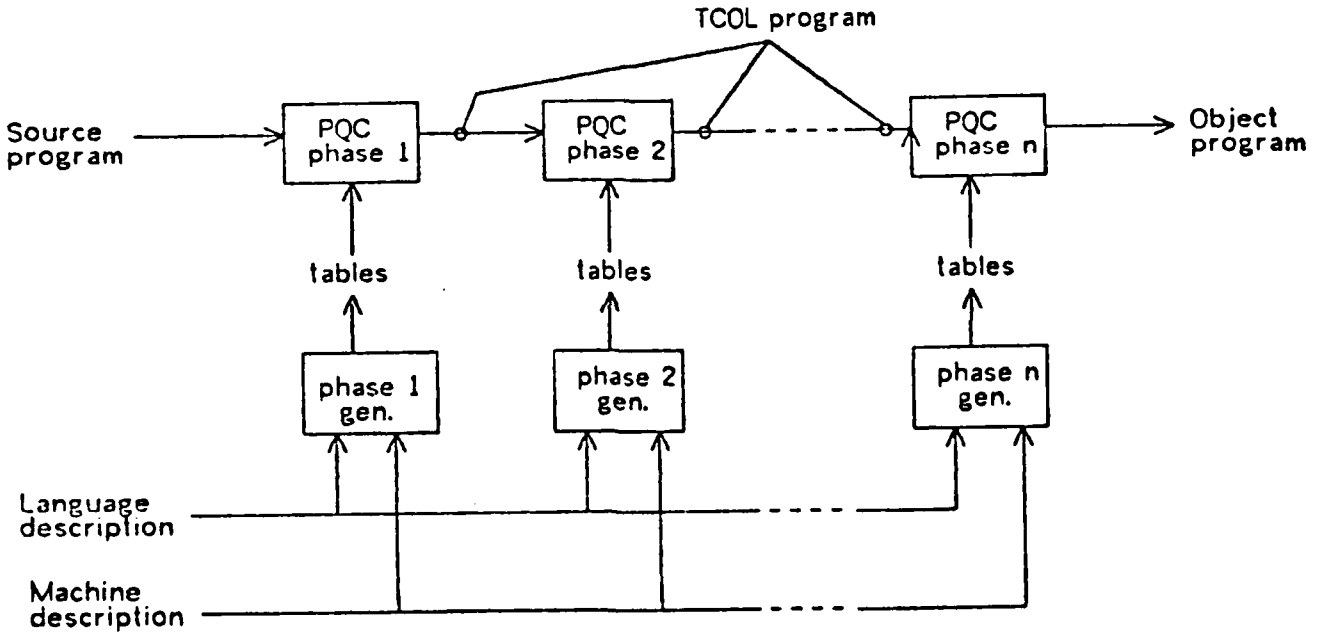


Figure 5-1: Detailed box diagram of PQCC system

The other major area of work was planning for the construction of an optimizing Ada compiler, a project to start in July 1979 and to continue for 18 months. While much of the retrospective analysis of the October compiler was clearly oriented towards this goal as well, several specific pieces of work were directed this way.

The TCOL.Ada paper describes a possible intermediate representation for Ada, the language designed to meet the Steelman requirements of the DOD standard language. This report has been written and is now being re-drafted because the Ada language has been chosen; we are able to make specific decisions about issues which could not be decided until Ada was chosen. Certain other details that were ignored initially will now be covered.

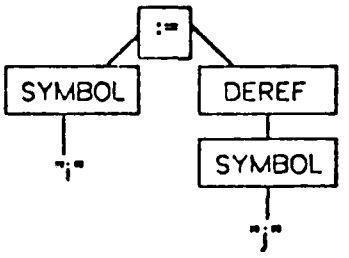
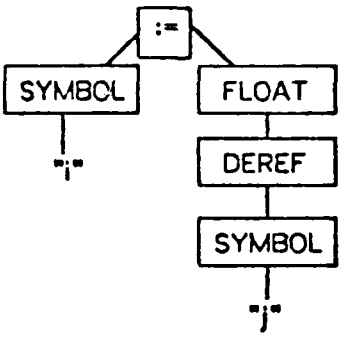
Statement	(Source Language)	TCOL tree
i = j i := j i ← .j	(Fortran) (Algol 68) (Bliss)	
a = j a := j or a := REAL(j)	(Fortran) (Algol 68)	

Figure 5-2: Examples of TCOL trees

Many of our support tools are being modified or re-written as we move from BLISS-10 to Common Bliss, the DEC supported and commercially distributed version of this implementation language. Common Bliss will allow us to write programs which can run on either our TOPS-10 PDP-10 systems or under VAX/VMS; the main goal of this is to enable us to utilize the VAX for development (the VAX systems will not be heavily used initially, while our PDP-10 is saturated). The PQC for Ada will be written in Common Bliss and will compile for either the PDP-10 (BLISS-36) or VAX (BLISS-32). Other support tools that were oriented towards a BLISS PQC compiler (e.g., the symbolic tree-printer) will be recoded in Common Bliss for an Ada PQC. This effort is currently under way.

In addition, we have been working with Intermetrics, who will be the subcontractor in the Ada effort. Intermetrics has already adopted, for their own use, several pieces of the PQCC technology; they are developing a language-independent Linear Graph LG support package for (initially) Pascal and PL/1, and are investigating the machine description technology of PQCC and the code generator-generator technology.

A new PQCC User's Manual is being produced, which will include all of the new features added to LG support since the previous edition. This manual now exists in draft form.

Ada projects

Members of this department (Wulf, Habermann, Shaw) have long been involved in the DOD High Order Language Commonality program which began in 1975 with the goal of establishing a single high order computer programming language appropriate for DOD embedded computer systems. An early effort here was the development of TARTAN, a language designed as an experiment to see whether the Ironman requirement for a common high order programming language could be satisfied by an extremely simple language. The results substantially met the requirements. Language description and low cost enhancements are described in the reports by Shaw et al.

Since the selection of the Green language for Ada, CMU has committed to two companion projects to support its other research efforts.

Pieces of the PQCC technology will be utilized to build an in-house Ada compiler, referred to as the Ada "charette"[1]. The compiler will eventually be written on the PDP-10's to generate code for a DEC VAX-11/780. The linear graph LG intermediate representation will be used to support the implementation of this compiler in Common Bliss. As indicated in the goals of this other project, it will not produce an optimizing compiler, but rather one which produces "conventional compiler quality" code. Certain simple optimization phases of the October PQCC may be usable in this compiler however. Code production is anticipated by Fall 1979.

We have also initiated a task which crosses the PQCC/Charette project boundary; it is to specify details of the runtime environment of Ada. This project, under Peter Hibbard and Paul Knueven two of the architects of the Algol-68 run time environment will investigate representation issues for stack frames, data structures, and other issues such that the run

---

[1] The *charette* was a small cart used during plagues to carry off the bodies of the dead. It later was used in the French national school of art to collect the projects as the time came to turn them in. The projects, which were feverishly and frantically worked on as their deadlines neared, were eventually called *charettes* by association.

time support will conform to the semantics of the Ada requirements.

Instruction set processor investigations

The work of Barbacci and Dietz in specifying, evaluating and validating computer architectures using their instruction set processor descriptions continues. Their primary tool is ISPS, a system which starts with a machine's ISP description, then compiles that to generate code for an artificial machine. The ISP simulator is then simply a software implementation of the artificial machine. The ISPS system was developed as part of the DARPA SMCD program. Their papers describe the application of these facilities to the Military Computer Family (MCF) and to the certification of architectures for the Computer Family Architecture (CFA) project.

5.3 Algol68

The Algol68 project has matured and is working toward its final stages. Some effort went into transferring the Algol68 system to UNIX, so that the investigations of the runtime system and the language can continue there.

One of the activities on the Algol68 system has been to build and incorporate an Adaptive Storage Allocator. The runtime system uses heap storage for all allocation of memory. This solves many of the memory management problems inherent in any language which declares and handles complex data objects, but its value is limited by the efficiency with which memory can be allocated and deallocated. The overheads of the memory allocator and deallocator, and the cost of storage fragmentation, have the biggest impact on this. An adaptive storage allocator tunes itself during the execution of a program to have optimal allocation and deallocation strategies, and to minimize fragmentation, for the particular memory request patterns which the program has. Analyses show that the technique has a significant impact on reducing distributed overheads and in avoiding unbounded housekeeping costs during execution. (Hibbard, Leverett in prep.)

The principle research activity has centered on the problem of linkage editing. Some background on this may be helpful.

A high-level language is implemented upon a "virtual machine" (VM): this machine has as its primitive operations the basic actions needed to support the language: parameter passing, procedure calls, assignments, etc. Its primitive data objects are those definable in the language: records, arrays, etc. Depending upon the level of the language the VM may be more of less close to the host computer hardware. For example, a low-level systems implementation language will have a VM which is more or less identical to the hardware,

supplemented by a few conventions about the placement of parameters, etc., and a high-level language will have a VM which is some distance from the hardware. The techniques for constructing the VM in software vary; furthermore several implementations of each VM may exist, each with the same external specification. They may be implemented by direct open-coding of the VM actions (which, in the case of an optimizing compiler may be done so abstrusely that the derivation of the compiler writer's possibly unconscious image of the VM may be difficult to abstract); they may be implemented by an interpreter acting upon some artificial instruction set; or they may be implemented by some intermediate technique.

In all cases, however, the VM design will impose some structure on the way that the physical resources of the host computer can be used; in particular on the way that memory addresses are allocated statically, when the program and its initial data are loaded, and dynamically while the program is executing. Even for the most machine oriented languages such conventions are used, and for high level languages assumptions as to the way the resources are used may be spread implicitly throughout the VM implementation, so that all other forms of access to the machine resources must be preempted, if the integrity of the language system is to survive. (Thus it may only be possible to access memory via the access mechanisms in the VM, since the positions of objects may be changed as a result of storage compaction.)

In order to have programs which comprise modules written in several different languages, it is necessary that the VMs of these languages are either compatible, or are able to coexist without interference. This either requires that conventions are imposed upon all language implementations, or that some languages are required to obey the conventions of the other languages. The first strategy has obvious disadvantages -- the design of the VM for a language is one of the most difficult and least structured parts of language implementation, and one in which large perturbations occur for seemingly trivial language features; the pre-specification of the features that all VMs must have hinder rather than assist in the construction of the VM. The second strategy similarly fails in the case of language systems which all impose their own strong brand of VM upon the host computer -- this is especially likely to be the case in languages intended for embedded applications, since fewer universal conventions are imposed by operating systems, etc.

Cases do exist, though, where such dissimilar VMs must be made to coexist -- Fortran routines called from a Lisp environment and vice versa, Algol68 called from Bliss and C, etc. For each such case either one builds an interface by hand, and defines, for each case, how the VMs shall coexist, or one cops out and uses gross features of the host machine, such as separate address-spaces for the VMs, separate processes for the modules in the different languages, and pipes or message to pass parameters between them, with a resultant loss of

efficiency and flexibility.

The linker which is now an integral part of the Algol68 system has been designed to explore these issues. It is, in fact, not specifically an Algol68 linker, or indeed any other language linker, but is more properly termed a multi-language linker. VMs may be defined as a part of the linking process; these are specified by defining the layout and properties of the segments of memory associated with each VM, and by libraries of low-level routines which support the primitive actions of the VM. Several different implementations of any particular VM may be used during a linking action: using in-line code expansion or out-of-line routines, by different representations of data objects, or with inbuilt monitoring actions, etc. The units of separate compilation are "modules" - the linking action causes the linker to insert the appropriate primitive action of the VM, using some specified implementation. Thus the linker acts in many respects like to code generator of a simple compiler; however, it delays the binding of the implementation of the primitive actions until as late as possible. This allows the linker to make more optimal decisions as to the placement of the code for these routines. Several VMs may be supported during linking; these form a nested hierarchy, allowing actions common to languages to be kept in common VMs. Additionally, since the linker is designed to produce code for a machine whose address space is potentially much smaller than its physical memory, it automatically handles the generation of overlays for code and data, thus allowing programs which are indefinitely large to be created.

5.4 Annotated Bibliography

Barbacci, M. R. An ISPS primer for the instruction set processor notation. In Computer Engineering: A DEC View of Hardware Systems Design. (C. G. Bell and J. C. Mudge and J. E. McNamara, Ed.) Digital Press, Bedford, MA, 1978. Also CMU-CSD Technical Report May 1979.

The ISPS computer description language is an evolutionary step toward the formalization of the digital design process at the higher or behavioral levels. ISPS is the second implementation of ISP as a computer language and has been used as a design tool which covers a wider area of application than any other hardware description language. Thus, besides simulation and synthesis of hardware, software generation, program verification, and architecture evaluation and control are amongst the current applications based on ISPS. The range of current and contemplated application areas are proof of the usefulness of the notation and its extension mechanisms. This paper is divided into two parts. The first part

describes the notation, its intended use, and the extension mechanisms which allow multiple applications or areas of research to co-exist and share machine descriptions. The second part briefly describes some of the current applications for ISPS; only enough detail is presented to illustrate the highly diverse set of problem areas that depend on a formal machine description. The appendix presents some language features not often found in programming languages (or machine description languages, for that matter). Topically, the appendix belongs after the first part. However, it is not critical to the understanding of the applications for ISPS and has been postponed to allow a smooth transition between the description of the notation and the description of its uses.

Barbacci, M. R., Dietz, W. and Szewerenko, L. Specification, evaluation, and validation of computer architectures using instruction set processor description. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, April, 1979.

Since early 1975, the Center for Tactical Computer Sciences (CENTACS) of the U.S. Army Electronics Command has been supporting an effort to develop a family of military computers based upon a common instruction set architecture. This Military Compute Family (MCF) will make available to DoD projects a series of computers which feature both an established software base and current hardware technology. The fundamental premise of the MCF project is that software compatibility should be achieved by the adoption of an existing, proven computer architecture for the MCF, thereby minimizing the risks inherent in the design of a new computer architecture and permitting the "capture" of an existing and evolving software base. In this context, computer architecture is distinguished from implementation considerations, and is defined as the structure of a computer which a machine level programmer needs to know in order to write any time independent programs which will run correctly on the computer. As part of this effort, Carnegie-Mellon is employing an ASP description and simulation facility for use in specifying, evaluating, and controlling these architectures. The overall goal is to provide the ability to precisely specify the selected architectures and to validate the correctness of hardware implementations.

Barbacci, M. R. Instruction set processor specifications for simulation, evaluation and synthesis. Proceedings of the 16th Design Automation Conference, San Diego, CA, June, 1979.

Formal descriptions of digital computers have been used traditionally for pedagogical purposes. Recent developments, however, illustrate that a formal computer description is highly useful in the simulation, evaluation, and synthesis of computers and other digital systems. In this paper we shall review the use of the ISP notation at Carnegie-Mellon University. The paper is organized into three sections that deal with the language, its use in the simulation and evaluation of computer architectures, and finally, its use in computer aided design. In this paper we emphasize the commonality between these seemingly disjointed applications. The reader is invited to consult the specific references for additional information on any particular subject.

Cattell, R. G. G. Using machine descriptions for automatic derivation of code generators. Proceedings of the Third Jerusalem Conference on Information Technology, Jerusalem, Israel, August, 1978. Also CMU-CSD Technical Report April 1978

Progress has been made in the design of compiler-compilers and translator writing systems, particularly with respect to automating the parsing of programming language text into internal notations. Much less progress has been made in automating the second part of the compilation process: translating the internal representation into instructions for the target machine. I believe this failure is primarily due to inadequate formalization of machines and the code generation process, rather than fundamental difficulties in automating the process. The goal of this paper is to study and formalize machines and code generators, and using these formalizations, to automatically derive code generators.

Cooprider, L. W. The representation of families of software systems. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, April, 1979.

Programming languages are notations for the representation of

algorithmic information, they are tools for "programming-in-the-small"[DeRe 76]. System description languages are notations for "programming-in-the-large". Because software systems often exist in several versions simultaneously, a system description language must accommodate parallel versions of systems and permit the natural expression of the information sharing among those versions. The construction of software systems involves sequences of construction processes such as text editing, compilation, document production, linkage editing, and cross-reference generation. Automation of these processes has been impeded by the use of inadequate models of software construction and maintenance. As a result, the enforcement of design decisions described in a system description language has been left to human agencies. We develop a notation for describing the subsystem interconnections of entire systems, the differences between versions of those systems, and the mechanisms by which the systems are constructed. Subsystems are objects which provide a set of resources to other subsystems and require a set of resources that are supplied by other subsystems. Each interconnection network can be instantiated in several versions. Versions are organized hierarchically so that similar versions share part of their descriptions. Detailed system construction processes, such as text editing, compilation and document generation, are expressed in a functional form. Resources and source files are combined according to construction rules to create the concrete objects that are the tangible (executable or readable) form of a software system. The construction processes are controlled by the interconnection structure and version specifications in which they are defined. This representation is the basis for the design of a software construction database. The database manager automatically performs system construction processes, propagates modifications to system components, and maintains construction histories. The database user can establish invariants in the database by attaching policies to each database object; the policies supply a set of actions to be performed when events in the database affect the object (e.g. a component has been modified). An extended example is presented to demonstrate the applicability of this representation to a real system. Several types of system

construction problems are discussed, and directions for improvements to the notation outlined.

Habermann, A. N. Implementation of regular path expressions. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, March, 1979.

Path expressions define sets of permissible operation sequences on typed objects. Path expressions specify process synchronization at a conceptual level instead of in terms of its implementation, which is the case if P,V operations or critical regions are hand-coded into the program text. A compiler takes care of translating path expressions into the necessary synchronization statements. This paper describes the compilation of regular path expressions which correspond to either deterministic or undeterministic finite state machines.

Leverett, B., Cattell, R. G., Hobbs, S., Newcomer, J., Reiner, A., Schatz, B. R. and Wulf, W. A. An overview of the production quality compiler-compiler project. Technical Report, Carnegie- Mellon University, Computer Science Department, Pittsburgh, PA, February, 1979.

The Production Quality Compiler-Compiler (PQCC) project is an investigation of the code generation process. The practical goal of the project is to build a truly automatic compiler-writing system. Compilers built with this system will be competitive in every respect with the best hand- generated compilers of today. They must generate highly optimized object code, and meet high standards of reliability and reasonable standards of performance. The system must operate from descriptions of both the source language and the target computer. Bringing up a new compiler, given a suitable language description and target architecture description, must be inexpensive and must not require the assistance of builders or maintainers of the compiler-writing system itself. This paper describes the goals and methodology of the PQCC project.

Oakley, J. D. Symbolic execution of formal machine descriptions. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, April, 1979.

As a tool having broad application to the growing area of automatically performing machine-dependent tasks, the symbolic execution of formal, procedural machine descriptions poses some difficult and fundamental problems not encountered in normal symbolic execution of algebraic languages. This thesis had two specific goals: (1) to identify the noteworthy problems of Machine Symbolic Execution (MSE) and to provide solutions for them, (2) to demonstrate, by an appropriate implementation, the feasibility of using the tool of MSE on a real life task - obtaining the input/output assertions for a machine's instruction set and creating from them what is called an assertion description. (This is a formal, non-procedural description of a machine, suitable as input to programs that require knowledge of the components of a target machine - instruction set, addressing modes, register complement, etc.). The research resulted in the following accomplishments: -A rigorous method was developed for symbolically evaluating algebraic-like expressions defined over the domain of variable-length bit-strings. A new, mixed integer/bit-string representation was devised which permitted the elimination of superfluous bit-lengths (by conversion to an integer-valued expression); non- superfluous bit-lengths were represented explicitly. A formal proof was given of the equivalence of this mixed representation with the original. Appropriate simplification techniques were introduced. -A technical definition of "instruction" was introduced, determined from the execution paths through the ISPS description. -A number of problems arising from commonly observed characteristics of real machines were resolved: optional independent execution of address-calculation routines was permitted (i.e., symbolically separate execution of operand- referencing and instruction-execution), including the possibility of certain types of side effects; the two most common mechanisms used on real machines for implementing variable-length instructions were incorporated; many issues were resolved concerning memory overlays, where multiple names can refer to the same parts of memory. -A program was written, in LISP, to symbolically execute a machine description and produce an assertion description as output. The output format and structure used was defined externally by an

existing research application for the automatic generation of code generators. The results above were all incorporated. The thesis describes a large example run on the symbolic executor, the DEC PDP-11 computer; appendices giving input and output are included. The work in this thesis was performed in the specific context of ISPS, a machine-readable successor to Bell and Newell's ISP language. The results obtained, however, have application to other machine description languages at the same general level of abstraction as ISPS. Finally, the thesis presents some conclusions on the use of the ISPS language for MSE.

Saunders, S. E. Compiling customized executable representations and interpreters. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, June, 1979.

This dissertation presents a view of the issues that arise in the design of executable representations--DELs-- for programs, and a compiler implementation which embodies both old and new responses to these issues. It focuses on methods of tailoring a DEL to fit a chosen combination of source language and host machine, and of customizing a tailored DEL to the specific needs and properties of particular source programs. An experimental compiler implementation incorporates several customizing methods, including a novel tree pattern search for specialized instruction synthesis. The measured performance of the implementation gives a promise that the methods presented will prove to be of value in the future. A tailored DEL provides better performance than more conventional "general-purpose" DELs (instruction sets) by taking account of the operation semantics, data types, name scope conventions, and control structures of a single source language or family of languages. The close match possible between source and DEL means that both the compiler and object code can be simpler, smaller, and faster. Customizing a DEL to a particular program takes advantage of ways in which the source fails to use the full generality of the language, or has different frequency distributions of usage from those assumed in the tailoring. A DEL can be customized by (1) omitting unused operations and facilities, (2) restricting ranges of values to those that actually appear, (3) adding new, specialized combination operations, (4) adjusting the encoding of instructions,

and (5) providing for encodings to vary between parts of the DEL program. The compiler must construct the DEL interpreter, or modify a kernel interpreter, to correctly handle each feature added or altered by the customizing. A customizing compiler and DEL were designed and implemented to test and demonstrate these ideas. XP compiles Pascal programs into a DEL intended for efficient interpretation on the Palyn EMMY computer. XP DEL instructions begin with a format code which specifies whether operands and result are on the stack or are explicitly named, in any combination. The completeness of the format set makes overhead instructions that simply move data to and from registers or the stack nearly unnecessary. The basic DEL code, simply customized by omitting unneeded opcodes and by restricting name fields (address offsets) and operation fields (opcodes) achieves a space performance slightly better than a carefully hand-tailored Pascal DEL on a range of sample programs, and much better than the PDP-10 Pascal compiler and Pascal "P". New instructions are synthesized and added to the interpreter based on patterns of operations and names in the parse-tree representation of the source program. The tree-pattern search algorithm uses depth-first search through the upper semi-lattice of patterns under the *subsumes* relation. That is, the search starts with patterns which match almost any piece of the tree and refine them recursively, guided by what actually appears in the program tree. The patterns found are ranked by an estimate of the profit that would result from installing each as a new operation. Customizing by searching for patterns in the tree leading to creation of specialized operations yields code 48 per cent to 60 per cent of the basic size, at moderate cost in interpreter growth and added compilation time. The object code sizes estimated by an independently developed compiler quality measurement procedure are respectively 0.64 and 0.36 times that of Bliss-11, a highly optimizing compiler.

Schatz, B. R., Leverett, B., Newcomer, J., Reiner, A. and Wulf, W. A. TCOLAda: An intermediate representation for the DOD standard programming language. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, March, 1979.

This document describes TCOLAda, a possible intermediate

representation for ADA -- the language being designed to meet the Steelman requirements. The purpose of this description is to provide enough information about this representation so that potential implementors of Ada can assess the feasibility of producing this representation at some intermediate stage of their compilers. Of course, Ada is not yet fully designed, and a choice between the two competing designs has not been made; thus, this document can only be considered tentative. It should, however, provide enough of the "flavor" of TCOLAda that an assessment can be made.

Shaw, M., Feldman, G., Fitzgerald, R., Hilfinger, P., Kimura, I., London, R. L., Rosenberg, J. and Wulf, W. A. Validating the utility of abstraction techniques. Proceedings of the ACM National Conference, Washington, DC, December, 1978.

A number of recent research efforts have been based on the hypothesis that encapsulation techniques, formal specification, and verification lead to significant improvements in program quality. As we gain experience with the language facilities produced by this research, we should attempt to validate that hypothesis. This paper poses this validation as the next major task in this area and outlines some ways to address it.

Shaw, M., Hilfinger, P. and Wulf, W. A. TARTAN language design for the ironman requirement: Notes and examples. ACM SIGPLAN Notices 13, 9 (September 1978). Also CMU-CSD Technical Report June 1978.

The Tartan language was designed as an experiment to see whether the Ironman requirement for a common high-order programming language could be satisfied by an extremely simple language. The result, Tartan, substantially meets the ironman requirement. We believe it is substantially simpler than the four designs that were done in the first phase of the DOD-1 effort. The language definition appears in a companion report; this report provides a more expository discussion of some of the language's features, some examples of its use, and a discussion of some facilities that could enhance the basic design at relatively little cost.

Shaw, M., Hilfinger, P. and Wulf, W. A. TARTAN language design for the ironman requirement: Reference manual. ACM SIGPLAN Notices 13, 9

(September 1978). Also CMU-CSD Technical Report June 1978.

Tartan is an experiment in language design. The goal was to determine whether a "simple" language could meet substantially all of the ironman requirement for a common high-order programming language. We undertook this experiment because we believed that all the designs done in the first phase of the DoD effort were too large and too complex. We saw that complexity as a serious failure of the designs; excess complexity in a programming language can interfere with its use, even to the extent that any beneficial properties are of little consequence. We wanted to find out whether the requirements inherently lead to such complexity or whether a substantially simpler language would suffice. Three ground rules drove the experiment. First, no more than two months would be devoted to the project. Second, the language would meet all the ironman requirements except for a few points at which it would anticipate Steelman requirements. Further, the language would contain no extra features unless they resulted in a simpler language. Third, simplicity would be the overriding objective. The resulting language, Tartan, is based on all available information, including the designs already produced. The language definition is presented here: a companion report provides an overview of the language, a number of examples, and more expository explanations of some of the language features. We believe that Tartan is a substantial improvement over the earlier designs, particularly in its simplicity. There is, of course, no objective measure of simplicity, but the syntax, the size of the definition, and the number of concepts required are all smaller in Tartan. Moreover, Tartan substantially meets all of the ironman requirement. (The exceptions lie in a few places where we anticipated Steelman requirements and where details are still missing from this report.) Thus, we believe that a simple language can meet the ironman requirement. Tartan is an existence proof of that. We must emphasize again that this effort is an experiment, not an attempt to compete with DoD contractors. Tartan is, however, an open challenge to the Phase II contractors: The language can be at least this simple!

6. System Architectures for Archival Memories

6.1 Introduction

Many advanced memory device techniques are evolving for the storage of extremely large data files. Most of these involve unique constraints, such as write once/read only. This research will evaluate the feasibility of alternative software and hardware architectures for interfacing these large memories to conventional computers and is a shared local network resource.

6.2 Current Research

To further understand the potential for a write-once read-only optical disk, CMU had several interactions with Philips Lab. Current capacity is 2×10^{10} bits per disk, with a potential for 10^{11} bits per disk. A carousel was described which might extend capacity to 10^{14} bits. The data rate with error correction is 5.24 Mbit/sec, with a maximum random access time of four seconds. Unfortunately, the cost is high ($200,000 for one read/write station and two read stations) and the availability is questionable (only five prototypes exist).

A strategy was developed for attacking the storage and retrieval problem. A set of design goals and a top level design was developed for a central file system for the CMU Computer Science Department. This file system will serve several purposes:

1. It will be a prototype of the multi-level file system needed for introduction of large archival memories.
2. It will be used for validation of archival system models.
3. It will provide a needed facility for our environment.

The basic design is for a transparent multi-level cache-like structure, with automatic migration between levels.

The principle objective is the completion of the design and initial implementation of the central file system. This will be the kind of file system architecture that currently appears necessary for graceful introduction of large archival memories. Since no such file system architecture currently exists, it is critical that we build one and understand its potentials and limitations. This objective is the key to work on the storage and retrieval problem.

The modeling problem, the memory hierarchy problem, and the system configuration problem all require a fairly complete data base of existing and proposed memory system characteristics and a complete survey of existing and proposed uses of large memory

systems. We intend to complete that data base and survey in fiscal year 1980.

Since optical memories appear to be the best near term technology for large archival memories, we intend to acquire one of the five Philips prototypes, interface it to the computer on which our central file system will be running, and attempt to integrate it into the central file system. The write-once characteristic and the limited lifetime of plastic disks (two years) make it unclear how such memories can best be used. We intend to explore several alternative strategies for their use.

A functional specification of the Archival Memories central file system has been completed. It will be published as a technical report with alternative design specifications by Summer 1980.

7. Signal Understanding in Distributed Systems

7.1 Introduction

Distributed sensor problems have always been an important aspect of defense problems. The technology now exists, however, to have widely distributed sensors share information and thereby function more effectively as an integral unit than was heretofore possible. CMU will be developing distributed software concepts that can operate successfully in a distributed sensor network.

7.2 Project Overview

This research will ultimately result in a concept demonstration system exploring the issues and problems of integrated distributed sensor networks (DSN). The operating scenario, or problem space, is for the system to locate, identify, and track in real time, a set of objects moving through a simulated terrestrial environment. An array of highly directional acoustic sensors (microphones) is used as the perceptual mechanism. The objects may vary in both type and number. Typical objects represent vehicles such as tanks, trucks, or trains. A complete set of situation displays is maintained for each area scanned by a sensor, integrated territory, and the entire domain.

The system is intended to run continuously and unattended. For these reasons the system is required to be highly fault tolerant, continuing to function even as a significant fraction of its components are intentionally disabled. Dynamic modification of both hardware and software is necessary for continuous operation and will be fully developed.

All programming is to be done in Ada, the DOD-adopted high level language, permitting an evaluation of the language for programming complex DOD applications and insuring development of a modular, easily modified software system.

7.3 Language and systems studies

The efforts expended in the last six months have concentrated in the areas of languages and systems suitable for the task domain. A complete specification for a DSN language is be being prepared and should be available as a technical report within six months. There is a requirement to extend Ada in order to make it usable for the specification of algorithms in distributed systems. Some of this analysis has been done for ALGOL68 and LO (see the Multiprocessor Systems section), but the mapping to Ada is not direct.

There is also a study underway on techniques for modifying large complex programs on a

distributed system without having to interrupt the entire system. The results of this work will provide techniques to permit dynamic modification and debugging of time critical, distributed function systems, of which DSN is only an example.

As a runtime system is being constructed for the Ada charette, consideration is being given in the design to its application to distributed systems. A prototype version should be operational by Fall 1980 for a VAX-11.

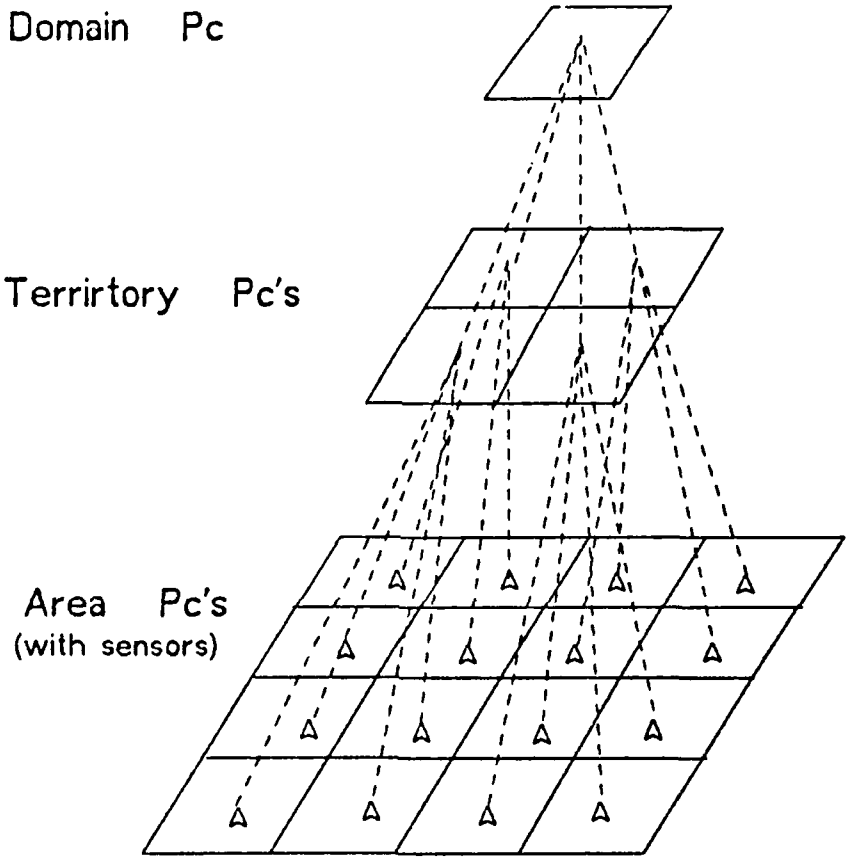


Figure 7-1: The logical structure of a pyramid sensor net architecture

7.4 Annotated Bibliography

Lesser, V. R. and Erman, L. D. An experiment in distributed interpretation. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, May, 1979.

The range of application areas to which distributed processing has

been applied effectively is limited. In order to extend this range, new models for organizing distributed systems must be developed. We present a new model, in which the distributed system is able to function effectively even though processing nodes have inconsistent and incomplete views of the data bases necessary for their computations. This model differs from conventional approaches in its emphasis on dealing with distribution- caused uncertainty and errors in control, data, and algorithm as an integral part of the network problem- solving process. We show how this new model can be applied to the problem of distributed interpretation. Experimental results with an actual interpretation system support these ideas.

8. Appendix: Publications

This is a cummulative listing of Computer Science Department publications from July 1, 1978 to June 30, 1979. They have been categorized according to area of research. The titles marked with an asterisk (*) indicate those publications whose abstracts are printed in this report.

Multiprocessing Systems

Fuller, S. and Harbison, S. The C.mmp multiprocessor. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, October, 1978.

Fuller, S. H., Ousterhout, J., Raskin, L., Rubinfeld, P., Sindhu, P. and Swan, R. J. Multi-microprocessors: An overview and working example. In Computer Engineering: A DEC View of Hardware Systems Design. (C. G. Bell, J. C. Mudge and J. E. McNamara, Ed.) Digital Press, Bedford, MA, 1978. Also CMU-CSD Technical Report 1977.*

Oleinick, P. The implementation and evaluation of parallel algorithms on C.mmp. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, November, 1978.

Raskin, L. Performance evaluation of multiple processor systems. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, August, 1978.

Siewiorek, D. P., Kini, V., Mashburn, H., McConnel, S. and Tsao, M. A case study of C.mmp, Cm*, and C.vmp--I. Experiences with fault tolerance in multiprocessor systems. Proceedings of the IEEE, Vol. 66, October, 1978.

Siewiorek, D. P., Kini, V., Joobani, R. and Bellis, H. A case study of C.mmp, Cm*, and C.vmp--II. Predicting and calibrating reliability of multiprocessor systems. Proceedings of the IEEE, Vol. 66, October, 1978. Also CMU-CSD Technical Report September, 1978.

Siewiorek, D. P., McConnel, S. R. and Tsao, M. The measurement and analysis of transient errors in Digital computer systems. Proceedings of the 1979 International Symposium on Fault- Tolerant Computing, Madison, WI, June, 1979. Also CMU-CSD Technical Report May, 1979.*

Swan, R. J. The switching structure and addressing architecture of an extensible multiprocessor: Cm*. Technical Report, Carnegie- Mellon University, Computer Science Department, Pittsburgh, PA, August, 1978.

Image Understanding Systems

Allen, G. R. and Juetten, P. G. SPARC - Symbolic processing algorithm research computer. Proceedings: Image Understanding Workshop. Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, November, 1978.

Eversole, W. L. et al. Investigation of VLSI technologies for image processing. Proceedings: Image Understanding Workshop. Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, November, 1978.

Kanade, T. A theory of origami world. Technical Report, Carnegie- Mellon University, Computer Science Department, Pittsburgh, PA, September, 1978.

Kender, John R. Shape From Texture: A brief overview and a new aggregation transform. Proceedings: Image Understanding Workshop. Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, November, 1978.

Reddy, D. R. Pragmatic aspects of machine vision. In Computer Vision Systems. (A. Hanson and E. Riseman, Ed.) Academic Press, New York, NY, 1978.*

Rubin, Steven M. The ARGOS image understanding system. Proceedings of the DARPA Workshop on Image Understanding, Pittsburgh, PA & Arlington, VA, November, 1978. Also CMU-CSD Technical Report, November, 1978.

Shamos, Michael I. Robust picture processing operators and their implementation as circuits. Proceedings: Image Understanding Workshop. Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, November, 1978.

Machine Intelligence

Berliner, H. J. The B* tree search algorithm; A best-first proof procedure. Artificial Intelligence 12 (1979).Also CMU-CSD Technical Report April, 1978.*

Carbonell, J. G. Counterplanning strategies: Computer models of human reasoning in conflict situations. Proceedings of the First International Symposium on Policy Analysis and Information Systems, Durham, NC, June, 1979. Also CMU-CSD Technical Report February, 1979.*

Forgy, C. L. On the efficient implementation of production systems. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, February, 1979.*

Gaschnig, J. Performance Measurement and Analysis of Certain Search Algorithms. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, May, 1979.*

Lesser, V. R. and Erman, L. D. An experiment in distributed interpretation. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, May, 1979.*

Lowerre, B. and Reddy, D. R. The Harpy speech understanding system. In Trends in Speech Recognition. (W. A. Lea, Ed.) Prentice- Hall, Inc., Englewood Cliffs, NJ, 1979. Also CMU-CSD Technical Report April, 1976.*

Mantei, M. M. and McCracken, D. L. Issue analysis with ZOG, a highly interactive man-machine interface. Proceedings of the First International Symposium on Policy Analysis and Information Systems, Durham, NC, June, 1979.*

McCracken, D. L. and Robertson, G. Editing tools for ZOG, a highly interactive man-machine interface. Proceedings of the International Conference on Communications, Boston, MA, June, 1979.*

McDermott, J. ANA: An assimilating and accommodating production system. Technical Report, Carnegie-Mellon University, Compute Science Department, Pittsburgh, PA, December, 1978.

Newell, A. Harpy, production systems and human cognition. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, September, 1978.

Oakley, J. D. Symbolic execution of formal machine descriptions. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, April, 1979.*

Rich, E. Building and exploiting user models. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, April, 1979.*

Robertson, G. Some design considerations for the ZOG man-computer interface. Proceedings of the Third NATO Advanced Study Institute in Information Science, Chania, Crete, Greece, August, 1978.*

Schatz, B. R. The computation of immediate texture discrimination. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, December, 1978.*

Yegnanarayana, B. Pole-zero decomposition of speech spectra. Technical Report,

Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, January, 1979.*

Yegnanarayana, B. and Ananthapadmanabha, T. V. Design of digital filters by pole-zero decomposition. Technical Report, Carnegie- Mellon University, Computer Science Department, Pittsburgh, PA, February, 1979.*

Yegnanarayana, B. and Ananthapadmanabha, T. V. On improving the reliability of cepstral pitch estimates. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, February, 1979.*

Yegnanarayana, B. Speech analysis by pole-zero decomposition. Preprint of Papers on Speech Communication, Acoustical Society of America, Cambridge, MA, June, 1979. Also CMU-CSD Technical Report September, 1978.*

Software Technology

Barbacci, M. R. An ISPS primer for the instruction set processor notation. In Computer Engineering: A DEC View of Hardware Systems Design. (C. G. Bell and J. C. Mudge and J. E. McNamara, Ed.) Digital Press, Bedford, MA, 1978. Also CMU-CSD Technical Report May, 1979.*

Barbacci, M. R. The symbolic manipulation of computer descriptions: An introduction to ISPS. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, August, 1978.

Barbacci, M. R., Dietz, W. and Szewerenko, L. Specification, evaluation, and validation of computer architectures using instruction set processor description. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, April, 1979.*

Barbacci, M. R. Instruction set processor specifications for simulation, evaluation and synthesis. Proceedings of the 16th Design Automation Conference, San Diego, CA, June, 1979.*

Cattell, R. G. G. Using machine descriptions for automatic derivation of code generators. Proceedings of the Third Jerusalem Conference on Information Technology, Jerusalem, Israel, August, 1978. Also CMU-CSD Technical Report April, 1978.*

Cooprider, L. W. The representation of families of software systems. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, April, 1979.*

Habermann, A. N. Implementation of regular path expressions. Technical Report,

Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, March, 1979.*

Jones, A. K. and Lipton, R. J. The enforcement of security policies for computation. Journal of Computer and System Science 17, 1 August, 1978.

Leverett, B., Cattell, R. G., Hobbs, S., Newcomer, J., Reiner, A., Schatz, B. R. and Wulf, W. A. An overview of the production quality compiler-compiler project. Technical Report, Carnegie- Mellon University, Computer Science Department, Pittsburgh, PA, February, 1979.*

Reid, B. K. SCRIBE introductory user's manual. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, August, 1978.

Saunders, S. E. Compiling customized executable representations and interpreters. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, June, 1979.*

Schatz, B. R., Leverett, B., Newcomer, J., Reiner, A. and Wulf, W. A. TCOLAda: An intermediate representation for the DOD standard programming language. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, March, 1979.*

Shaw, M., Feldman, G., Fitzgerald, R., Hilfinger, P., Kimura, I., London, R. L., Rosenberg, J. and Wulf, W. A. Validating the utility of abstraction techniques. Proceedings of the ACM National Conference, Washington, DC, December, 1978.*

Shaw, M., Hilfinger, P. and Wulf, W. A. TARTAN language design for the ironman requirement: Notes and examples. ACM SIGPLAN Notices 13, 9 (September 1978). Also CMU-CSD Technical Report June, 1978.*

Shaw, M., Hilfinger, P. and Wulf, W. A. TARTAN language design for the ironman requirement: Reference manual. ACM SIGPLAN Notices 13, 9 (September 1978). Also CMU-CSD Technical Report June, 1978.*

System Architectures for Archival Memories

Barbacci, M. R. and Sproull, R. F. System organizations for archival memories. Proceedings: Workshop on Archival Memory Technology. Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, September, 1978.

Sproull, R. F. and Barbacci, M. R. File system strategies for archival memories. Proceedings:

Workshop on Archival Memory Technology. Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, September, 1978.

Distributed Sensor Networks

Habermann, A. N. Dynamically modifiable distributed systems. Proceedings: Workshop on Distributed Sensor Nets. Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, December, 1978.

Hibbard, Peter G. Language design for distributed systems. Proceedings: Workshop on Distributed Sensor Nets. Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, December, 1978.

Sproull, R. F. and Cohen, Dan High level protocols. Proceedings: Workshop on Distributed Sensor Nets. Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, December, 1978.